LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no. 824-829
cop. 2

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

OCT 19 1977

L161—O-1096

Il6r
no.829
Cop 2

UIUCDCS-R-76-829

Math

COO-2383-0036

8

A User's View of Solving Stiff Ordinary Differential Equations

by

L. F. Shampine
C. W. Gear

September 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

The Library of the
FEB 03 1977
University of Illinois
at Urbana-Champaign

Digitized by the Internet Archive
in 2013

http://archive.org/details/usersviewofsolvi829sham

UIUCDCS-R-76-829 COO-2383-0036

A User's View of Solving Stiff Ordinary Differential Equations

by

L. F. Shampine*
C. W. Gear

September 1976

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* Numerical Mathematics Division, 5122, Sandia Laboratories, Albuquerque, New Mexico 87115.

Supported in part by the Energy Research and Development Administration under contract US ERDA AT(11-1) 2383.

ABSTRACT

This paper aims to assist the person who needs to solve stiff ordinary differential equations.

First we identify the problem area and the basic difficulty by responding to some fundamental questions: Why is it worthwhile to distinguish a special class of problems termed "stiff"? What are stiff problems? Where do they arise? How can we recognize them?

Second we describe the characteristics shared by methods for the numerical solution of stiff problems. These characteristics have important implications as to the convenience and efficiency of solution of even routine problems. Understanding them is indispensable to the assembling of codes for the very efficient solution of special problems or for solving exceptionally large problems at all.

Third we shall briefly discuss what is meant by "solving" a differential equation numerically and what might be reasonably expected in the case of stiff problems.

1. INTRODUCTION

The numerical solution of ordinary differential equations is an old topic and, perhaps surprisingly, methods discovered around the turn of the century are still the basis of the most effective, widely used codes for this purpose [1]. Great improvements in efficiency have been made, but it is probably fair to say that the most significant achievements have been in reliability, convenience, and diagnostic capabilities. The typical scientific problem can be solved by casual users of these codes both easily and cheaply. Nevertheless, there are several kinds of problems which classical methods do not handle very efficiently. The problems called "stiff" are too important to ignore, and are too expensive to overpower. They are too important to ignore because they occur in many physically important situations. They are too expensive to overpower because of their size and the inherent difficulty they present to classical methods, no matter how great an improvement in computer capacity becomes available. Even if one can bear the expense, classical methods of solution require so many steps that round-off errors may invalidate the solution. It is all the more frustrating that the solutions of stiff problems look like they ought to be particularly easy to compute. After a few general remarks about solving differential equations we shall use some simple examples to show where the trouble originates and what might be done about it. There are effective codes available for the solution of stiff problems, but it is necessary that the user have some idea how they work in order to take full advantage of them. Lastly we discuss what are realistic goals when solving a stiff differential equation.

2. WHAT ARE STIFF PROBLEMS?

When solving

$$y' = f(x, y) \quad , \quad y(0) \text{ given} , \tag{2.1}$$

we must consider the behavior of solutions near to the one we seek. This is because as we step along from $y_n \doteq y(x_n)$ to y_{n+1} approximating $y(x_n + h)$ we make inevitable errors causing us to move from the desired integral curve to a nearby one. If we make no further errors, we follow this new curve so that the resulting error depends on the relative behavior of the two solution curves. Let us consider the example

$$y' = A(y - p(x)) + p'(x) \quad , \quad y(0) = v , \tag{2.2}$$

where A is a constant. The analytical solution is

$$y(x) = (v - p(0)) \exp(Ax) + p(x) . \tag{2.3}$$

If A is large and positive, the solution curves for the various v fan out and we say the problem is *unstable*. Such a problem obviously is difficult for any general numerical method which proceeds in a step by step fashion. When A is small in magnitude the curves are more-or-less parallel and such *neutrally stable* problems are easily handled by conventional means. When A is large and negative, the solution curves converge very quickly. In fact, whatever the value y(0), the solution curve is virtually identical to the particular solution p(x) after a short distance called an *initial transient*. This super-stable situation is ideal for the propagation of error in the differential equation but not, as it turns out, for the propagation of error in a numerical scheme. The last class of problems is called *stiff*.

Equation (2.2) is of more general significance than its special form suggests. The behavior of solutions of (2.1) near a particular solution g(x) can be studied by linearizing the system of equations (2.2) as

$$y' \doteq J(x, g(x))\,(y - g(x)) + f(x, g(x))$$
$$= J(x, g(x))\,(y - g(x)) + g'(x)$$

where the Jacobian $J = \partial f/\partial y$. This approximation can be justified in a limiting argument, but we are only going to use it qualitatively and will not attempt rigor. If we further suppose that the Jacobian is slowly varying, we can approximate it locally by a constant matrix. After a principal axis transformation these equations are uncoupled into a set of equations each of the form (2.2). In this general situation A is an eigenvalue of the Jacobian; hence it may be a complex number. By a stiff problem we mean one which is very stable and for which the Jacobian is slowly varying along the solution curve. The stability requires that no component be unstable (no eigenvalue has a real part which is at all large and positive) and that some component be very stable (some eigenvalue has a real part which is large and negative). Further we suppose that the limit solution is slowly varying, for if it is not, then classical methods work as well as one might hope for. Thus one tries to solve problems which may have a difficult transient but thereafter the solution is slowly varying and the error propagation behavior of the differential equation itself is extremely good.

To expose the trouble let us integrate (2.2) by Euler's method when A is a large negative number and p(x) is slowly varying. This scheme advances from y_n to an approximation at $x_n + h_n = x_{n+1}$ by $y_{n+1} = y_n + h_n f(x_n, y_n) = y_n + h_n y_n'$. This being the linear terms of a Taylor series expansion at x_n, the local truncation error of the method is $h^2 y''(x_n)/2 + O(h_n^3)$. A code which selects its step size so that the local truncation error is approximately ε (as most codes do) will choose

h so that $\varepsilon \doteq |h_n^2 y''(x_n)/2|$. If the numerical solution is close to the true solution, we can deduce the behavior of h from

$$y''(x) = (v - p(0))A^2 \exp(Ax) + p''(x) \quad .$$

When x is small, it is clear that

$$h_n \doteq \left(\frac{2\varepsilon}{|y''(x_n)|} \right)^{1/2} \doteq \left(\frac{2\varepsilon}{|(v - p(0))A^2|} \right)^{1/2} \quad .$$

When x is large the exponential term disappears so that

$$h_n \doteq \left(\frac{2\varepsilon}{|p''(x)|} \right)^{1/2} \quad .$$

By assumption $|A|$ is large and $|p''(x)|$ is small so that we have quantified the statement that the step size needed for accuracy must initially be small to resolve the rapid change of the transient but eventually becomes large and independent of A.

This is not the whole story. There are two main factors affecting the size of the step -- accuracy and stability. Accuracy refers to smallness of the local error, that is, the error introduced in a single step. Stability refers to errors not growing in subsequent steps. We have seen for this example that accuracy is easily handled. Let us now examine stability. Because this is a linear differential equation and a linear method, it is easy to solve for the global error which is the difference between the numerical solution at any point and the true solution. If we define the global error as

$$\delta_n = y_n - y(x_n) \quad ,$$

we find that

$$\delta_{n+1} = (1 + h_n A)\delta_n + [y(x_n) + h_n y'(x_n) - y(x_{n+1})] \quad .$$

This says that the global error after the n-th step consists of the error

propagated from the previous point x_n plus the local truncation error in the n-th step. This error is amplified unless $-2 \leq h_n A \leq 0$. Clearly this restriction dominates in the selection of the step size once outside of the transient region. Note that a problem is not stiff in the transient region because $h_n A$ must be small to control the local truncation error.

The essence of the matter is that for most problems the accuracy requirement dictates the choice of step size, but for some, the stiff problems, the stability requirement does. In general we must discuss the stability of the difference scheme when A is a complex number. Analyzing stability as we did with Euler's method, we now find a region in the left half complex plane, called the region of absolute stability, in which $h_n A$ must lie for the difference scheme to be stable. For Euler's method this region is the disc of radius 1 centered at (-1, 0). Though the approximations are crude, this analysis does furnish a good qualitative understanding of the local behavior of the difference schemes. The stability restriction takes the form that $|h_n A|$ not be too large. As a practical matter this is no restriction unless $|A|$ is "large" and the accuracy requirement is easy to meet.

One worry should be dispelled at once. When implemented properly, the instability on encountering stiffness of classical methods such as Euler's is automatically detected and handled by reducing the step size. [2, 3]. Computer programs suitable for non-stiff problems do not "blow up" in the presence of stiffness, they just become inefficient. The reason for this is easy to see for methods like Euler's. Automatic codes estimate the local error by estimating a derivative of the solution. The only way in which this can be done economically involves applying some form of

difference operator to the computed solution. With the Euler method, for example, we could form the second difference of the solution to estimate the second derivative. For the linear example discussed, the second difference will consist of two parts; the second difference of the true solution and the second difference of the global error. A simple calculation for constant step size h shows that the latter is $(h\,A)^2\,\delta_{n-1} + h^3(A\,y''_{n-1} + y'''_{n-1})/2$. If h A is large, this term dominates, so the step control mechanism will reduce the step size accordingly.

The problem of instability for large step sizes when solving stiff equations is common to all methods that are efficient for non-stiff equations. All methods give rise to a global error equation of the form

$$\delta_{n+1} = S_n\,\delta_n + \varepsilon_n \quad ,$$

where ε_n is the local truncation error and S_n is the error amplification matrix whose size depends on the Jacobian of the differential equations. Methods for non-stiff problems are chosen so as to make the local truncation error term ε_n small for best efficiency. When stiff problems are to be solved, it is necessary to sacrifice some of the accuracy in order to improve the stability. Methods suitable for stiff equations are such that S_n is small for Jacobians with large negative eigenvalues. The example of the backward Euler method

$$y_{n+1} = y_n + h\,y'_{n+1} = y_n + h\,f(x_{n+1},\, y_{n+1})$$

is informative because it is easy to analyze and yet typical of many methods for stiff equations. When it is applied to equation (2.2) we get a global error equation of the form

$$\delta_{n+1} = \delta_n + h_n\,A\,\delta_{n+1} + [y(x_n) - y(x_{n+1}) + h_n\,y'(x_{n+1})]$$

or

$$\delta_{n+1} = (1 - h_n A)^{-1} \delta_n + (1 - h_n A)^{-1} h^2 y''(x_n)/2 + O(h^3)$$

The propagated error is damped whenever $|1/(1 - h_n A)| \leq 1$ -- which includes the whole left half plane. This is a marvelous improvement since an apparently minor change in the scheme has completely done away with a stability limitation. However, the backward Euler scheme is implicit, meaning that a nonlinear equation must be solved at each time step to determine y_{n+1}. This is characteristic of methods with very good stability properties and we shall examine the cost later. For now we just comment that because the stability requirement is so much more stringent for the forward Euler method than the accuracy requirement, the backward Euler method permits orders of magnitude improvement in efficiency for typical stiff problems even though each step is much more expensive.

The essence of stiffness is that one has a very stable integral curve which one wishes to compute efficiently when it is changing slowly. Most physical systems of interest are going to be stable; those which permit very rapid change are the ones which are potentially stiff. Thus one should be alert to physical components with greatly different time constants. For example, control systems are intended to provide stability. When they very quickly correct a deviation from a desired slowly changing path, the differential equations describing them will be stiff.

It is important to appreciate that stiffness depends on the behavior of all nearby solutions, that is, on the differential equation rather than the behavior of the solution itself. For example, the equation

$$y' = (v - p(0))A \exp(Ax) + p'(x) \quad , \quad y(0) \text{ given}$$

has exactly the same solution as (2.2) when both have the initial value v, but is not stiff at all. For this quadrature problem the integral curves are parallel and the problem is of neutral stability. Thus the presence of some components which change at a rate much faster than others is not necessarily an indication of stiffness. Still, physical systems are usually stable so this is a pretty reliable indication of stiffness in the proper context. As examples, chemical reactions with large rate constants and nuclear reactions with species decaying at rates varying widely typically lead to stiff equations. Electrical circuitry involving fast elements such as high speed transistor models ordinarily lead to stiff problems. A survey of such applications with examples can be found in [4]. Rather than take up many examples, we shall look in some detail at the use of semi-discretization to solve a simple partial differential equation. The resulting stiff system of equations will illustrate a number of points we shall need later.

Suppose we want to solve the heat equation

$$\frac{\partial y(x, t)}{\partial t} = \frac{\partial^2 y(x, t)}{\partial x^2}$$

$$y(0, t) \equiv 0,\ y(1, t) \equiv 0,\ y(x, 0) \text{ given}$$

Let $\Delta x = 1/N$ and $x_i = i\ \Delta x$ for $i = 0, 1, \ldots, N$. Suppose that $y_i(t)$ is to approximate $y(x_i, t)$ where $y_i(t)$ arises from replacing the space derivative in the heat equation by a centered difference:

$$\frac{dy_i(t)}{dt} = \frac{1}{(\Delta x)^2}\left(y_{i+1}(t) - 2y_i(t) + y_{i+1}(t)\right) \qquad i = 1, \ldots, N - 1$$

$$y_0(t) \equiv 0,\ y_N(t) \equiv 0,\ y_i(0) = y(x_i, 0) \text{ each } i$$

One easily finds the constant Jacobian and that its eigenvalues are

$$A_{N-n} = - \left[\frac{\sin(n \pi \Delta x/2)}{\Delta x/2} \right]^2$$

from which we see that

$$A_1 \doteq \frac{-4}{(\Delta x)^2} \quad , \quad A_{N-1} \doteq - \pi^2$$

If Euler's method is used to solve the set of ordinary differential equations with step size Δt, we conclude that for stability $|h A_1| \leq 2$ or

$$\frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

Here how stiff the problem is depends on how fine a spacial mesh we choose initially. If N is small, one is well advised to use non-stiff methods, but for large N this is not economic. Note that if we proceed in this way, we are using the classical (fully discrete) forward difference scheme for solving the heat equation. The backward Euler method corresponds to the classical backward difference scheme and, as is well known, there is then no limitation on Δt for reasons of stability. Considerable experience on the part of people interested solely in partial differential equations shows that it is much cheaper to use the more stable method with the more expensive steps. Experience applying general purpose codes for ordinary differential equations agrees with this and adds the observation that enhanced efficiency and reliability can be obtained by variation of step size and formula.

It should be noted that the eigenvalues obtained in this example are not due solely to the spatial discretization used. The original partial differential equation has eigenvalues of $- (k \pi)^2$, $k = 1, 2, \ldots$ so the first eigenvalue of the discretized system is approximately the first eigenvalue of the differential operator, and the others are approximations

to some of the larger ones. This points out that the stiffness is inherent in the problem, not part of the method of solution. Either the problem (that is, the model of the physical situation) must be changed, or stiffness must be faced in the solution process.

In a number of areas, particularly chemical kinetics, problem solvers have removed stiffness by changing the model. The idea is that physical considerations allow some components to be recognized as changing on time scales much shorter than those of other components. Think, for example, of a chemical reaction taking place in a moving medium or the rolling of a rocket as compared to its motion along its trajectory. Quasi-static or pseudo steady-state approximations hold one set of components fixed in value over suitable time periods either because they change so slowly that changes can be neglected or because they change so rapidly that steady-state values are achieved almost instantly. Such approximations lead to sets of algebraic equations coupled to (hopefully non-stiff) sets of ordinary differential equations. In some cases approaches of this kind have worked very well, but it is hard to relate the solution of the modified model to that of the original model. We shall reconsider this technique in the next section where it will be seen that the methods for stiff problems do much the same thing automatically. Current codes for stiff differential equations are sufficiently efficient that there is no need to consider such model changes for most problems for reasons of cost, and there are excellent reasons of convenience and theoretical support for not changing the model.

The example of the partial differential equation shows how large systems can arise. (Consider the situation with several space variables.)

Both it and the remarks we have made so far about steady-state approximations show that stiffness is not something unfamiliar; it has arisen, been studied, and dealt with, in other contexts. The example also exhibits the limited coupling that is usually present in large systems -- here each equation involves only three unknowns or fewer, regardless of the number of equations in the system. Exploitation of this property will be mentioned in the fourth section. It is an important aspect of efficiency.

We have touched on several ways of realizing stiffness is present. Before discussing solution methods let us summarize them. Often one has a considerable understanding of the qualitative behavior of solutions of (2.1) on physical grounds. If the system is known to be very stable, it is likely to be stiff. If some variables are known to change on time scales very different from others and the physical problem is well posed, the governing equations are likely to be stiff. Analysis or experience with similar or model problems is often very useful. A common sign of stiffness is that a code aimed at non-stiff problems proves conspicuously inefficient for no obvious reason (such as a severe lack of smoothness in the equation or the presence of singularities). There are codes for non-stiff problems [3, 5] which rather reliably diagnose stiffness automatically.

The integration during the transient has the step size limited by accuracy rather than considerations of stability so this part of a differential equation problem is not stiff. The distinction might be made more vivid by a little anecdote. The authors recently participated in a conference on the solution of stiff differential equations arising in models of lasers. One speaker discussed a refined physical model involving

some 250 energy levels which was consuming hours of computer time. Though models with these states aggregated were clearly stiff, his problem was being solved very inefficiently by codes aimed at stiff problems. Indeed, he found codes aimed at non-stiff problems to be conspicuously more efficient. Each solution component had the same qualitative behavior. After a while the level would become populated and the population would rapidly grow to a number about which it varied slowly. The difficulty originates in the fact that the various levels are populated successively and there are a great many of them. As far as the codes are concerned, they are always working on the transient for some energy level. Eventually, of course, the whole system would get into a steady state and stiff methods would show their worth, but the cost of reaching this was prohibitive. The non-stiff methods perform better in the transient because this is what they are designed to do. Besides illustrating the role of transients, this example also points out that we do not have codes, or even methods, capable of adequately solving all the problems of scientific interest in the area of differential equations.

3. CHARACTERISTICS OF SOLUTION METHODS

All of the methods used in general purpose codes for the solution of stiff differential equations are implicit of necessity. This means that an equation must be solved at each step to obtain the numerical approximation. For example, in the backward Euler method,

$$y_{n+1} = y_n + h_n f(x_{n+1}, y_{n+1})$$

must be solved for y_{n+1}. If the problem is linear (that is, if f is linear), then a linear equation must be solved, but if the problem is nonlinear, a nonlinear equation must be solved. Simple functional iteration is used in codes for non-stiff problems to solve such equations:

$$y_{n+1}^{(m+1)} = y_n + h_n f(x_{n+1}, y_{n+1}^{(m)})$$

For the model problem (2.2) we easily find that the iteration error satisfies

$$y_{n+1}^{(m+1)} - y_{n+1} = h_n A(y_{n+1}^{(m)} - y_{n+1})$$

Once again we encounter the effects of stiffness -- this simple iteration will not converge if we have a stiff problem (for which $|h_n A| > 1$). The usual procedure for stiff problems is some variant of Newton's method. This uses an approximate Jacobian J and one solves repeatedly the linearized system

$$y_{n+1}^{(m+1)} = y_n + h_n f(x_{n+1}, y_{n+1}^{(m)}) + h_n J(y_{n+1}^{(m+1)} - y_{n+1}^{(m)})$$

For the model problem (2.2) the iteration error satisfies

$$y_{n+1}^{(m+1)} - y_{n+1} = (1 - h_n J)^{-1} (h_n A - h_n J) (y_{n+1}^{(m)} - y_{n+1}) \qquad (3.1)$$

Any reasonable approximation J to A will cause this iteration to converge. If the problem is not stiff so that $h_n A$ is small, all that matters is that $h_n J$ be small. If the problem is stiff so that $h_n A$ is large and

negative, all that matters is that $h_n J$ be within about a factor of 50% of $h_n A$. For this example, and for any linear problem, the iteration converges in one step if J is exactly A.

This analysis extends to systems, in which case the iteration (3.1) involves solving a linear equation at each iterate. If the problem is linear, only one iteration is necessary (assuming that J = A). In the case of non-linear problems, more than one iteration may be necessary. In fact, in problems arising from the spatial discretization of partial differential equations, one of the biggest concerns is the solution of the nonlinear system at each step. This need not be a cause of concern if a good starting approximation is obtained, and the situation is such that a good starting approximation is almost always available by use of an explicit integration formula, usually called a predictor. Although a predictor is an integration formula, it does not have any of the main costs associated with a formula suitable for stiff problems because it is really a polynomial extrapolation process using information already known about the solution. For example, if the backward Euler formula is being used to solve stiff equations, the forward Euler formula can be used as a predictor. The forward Euler formula uses the function value y_n and the derivative y'_n computed in the last step to estimate the value of y_{n+1} to be used as the first iterate $y^{(0)}_{n+1}$. The accuracy of the predictor is as good, and usually better than that of the actual integrator (called the corrector) for stiff problems. The purpose of the corrector is to provide stability. When this technique is used along with a number of other techniques to detect convergence quickly, an average of less than 1.5 iterations of equation (3.1) are needed at each step in typical problems.

(Using a predictor has several other advantages. In particular, the difference between the predictor and corrector provides a reasonable error estimator for local truncation error -- an important part of any code. See Gear [6].)

It is worth noting that convergence of a quasi-Newton method for the corrector iteration is guaranteed for small enough step sizes because as the step size is reduced, an iteration such as (3.1) becomes a contraction. In addition, the accuracy of the predictor improves as h is reduced so that the initial approximation is more likely to be in the region of convergence.

The linearity of a problem does little to reduce the solution time in current codes for stiff problems. This could indicate a need for the development of better methods for linear problems, but it seems more likely that the reason is the effects of nonlinearity are being handled very efficiently.

Returning now to the idea of pseudo steady-state approximations, we suppose that the equations can be written in the form

$$y' = f(x, y, z) \quad , \quad \varepsilon z' = g(x, y, z) \tag{3.2}$$

where the solution components are split into two groups y and z. Except in the transient, or boundary layer as commonly termed in this context, the small parameter ε suggests one neglect the term $\varepsilon z'$ and solve instead the algebraic-differential system

$$y' = f(x, y, z) \quad , \quad 0 = g(x, y, z) \tag{3.3}$$

In a number of significant physical applications this kind of approximation results in an easy (that is, non-stiff) set of ordinary differential equations and a set of algebraic equations. Even in this favorable

situation it is not clear how to assess the errors which arise. Most often one has to solve the algebraic system by numerical means. If a code for stiff equations is applied to equations (3.2) directly, it effectively solves equations (3.3) in the stiff region. This can be seen by considering the application of the backward Euler method to the second of equations (3.2) when $\varepsilon z'$ is small. We get

$$\frac{\varepsilon}{h_n}(z_{n+1} - z_n) = g(x_{n+1}, y_{n+1}, z_{n+1}) \doteq 0$$

The fact that a code for stiff equations is automatically doing much the same thing as an analyst might, indicates the power of the kind of schemes we are discussing and shows how well-conceived and well-executed software can provide the casual problem solver with great assistance.

4. CODES FOR STIFF PROBLEMS

The backward Euler method, and similar methods such as the trapezoidal rule, were the first discovered which could handle stiff problems reasonably well so they became widely used. Because the large codes written to facilitate specific application areas such as simulation, chemical kinetics, circuit analysis, and the like are very slow to change, these methods are still seen in practice. More efficient methods are now available as a result of researches into higher order methods. The initial transients alone represent difficult non-stiff problems and the great success of high order methods for non-stiff problems has encouraged such investigations. There has been no lack of ideas for high order procedures with excellent stability but improvements have not been won easily because each seems to raise some new difficulty. Relatively few ideas have been implemented as software of sufficiently high quality to merit general consideration. We shall mention a few such ideas in order to allude to some of the difficulties later. Also, we shall mention specific codes because there are so few general purpose codes for stiff problems which are widely available. Because in some cases we do not have experience with the codes we are not necessarily endorsing them; we do have reason to believe that all are serious attempts at providing mathematical software for this problem so we hope that learning of them will prove useful to the reader.

Using the general idea of extrapolation [7] there is a code of Schryer [8] which does repeated extrapolation of the backward Euler formula. In doing this one adapts the order to the requirements of the problem. Lindberg [9] has written a code which extrapolates the midpoint rule a single time to generate a method of order four. The

popular Runge-Kutta methods furnish methods suitable for stiff equations if one considers implicit methods. Hulme [10] has written a code which provides an arsenal of implicit Runge-Kutta methods; there are two families of methods each with a large range of orders. Though a fixed formula is used for the integration, the code can select an appropriate one automatically. The most popular formulas being used in general purpose codes are the backward differentiation formulas (see Gear [11]). Like their relatives the Adams formulas, these formulas are usually implemented so as to automatically choose the step size and the order. An early code is that of Gear [12]. It has been often reprogrammed and various improvements made; a generally available and widely used version is that of Hindmarsh [13]. Extensions which are reported to improve the efficiency of the methods are given by Skeel and Kong [14].

The backward differentiation formulas illustrate an important point about codes for stiff problems. As the stability plots in [15] show, if the Jacobian has eigenvalues near the imaginary axis, the higher order formulas will be unstable. The unstable region increases as the order does to the point that formulas of order greater than six are not stable at all. Most codes do not use the sixth order formula because of this limitation though it is stable in most of the left half plane. The formulas of order five and lower do not have much of a limitation of this kind but occasionally one notices it with a real problem. Just because one has a good code implementing a good method does not mean that he will not encounter difficulties with stiffness. One needs some understanding of the characteristics of his code and it is best to have a repertoire of methods on which to draw.

The implicit Runge-Kutta methods avoid the problems associated with the high-order backward differentiation formulas because they can be stable in the whole left half plane. However, there is a price associated with this additional stability. The system of equations which must be solved is two or more times larger, implying a large increase in storage for large systems. When these methods are effective, it is because they can use a much longer step than methods based on backward differentiation. Sometimes, however, the solution will change character as we move from a smooth stiff region back into a rapidly varying transient within one long step which is then wasted. This same kind of difficulty affects the efficiency of extrapolation methods which gain their speed from a very long basic step.

A good code selects its step size, and hence x_n, automatically. If the user is prepared to accept output at only the points x_n, there is no impact on the efficiency of the integration. If, however, the user must compute the solution at many other points, the cost can become prohibitive. If there are a sufficient number of integration points that the desired accuracy can be achieved by interpolation, the cost is small and there is no effect on the integrator. In fact, most codes based on the backward differentiation formulas have scaled derivatives or divided differences available so that the interpolation cost is minimal; the better codes provide interpolation subroutines for the user. If, on the other hand, the code must be asked to compute intermediate points by integration, the cost can be high, particularly in codes which use long steps such as implicit Runge-Kutta and especially in extrapolation.

We see that the solution of stiff problems is practical provided we are prepared to solve at each step a linear system based on the

Jacobian. This can have serious consequences. The function f has n components but in general its Jacobian has n^2 components. For a general problem of even moderate size, providing the Jacobian analytically can be burdensome for the programmer and it can be very difficult to be sure that the partial derivatives have been obtained correctly. For these reasons, automatic generation of a Jacobian by numerical differencing is an indispensable part of a general purpose code.

The evaluation of Jacobians is a relatively expensive part of the solution process even though good codes try to do this as infrequently as possible. The key to reducing this cost is to take advantage of special structure of the problem. Ordinarily, medium to large systems have weak coupling so that most partial derivatives are zero; linear terms in the equations giving rise to immediately available partial derivatives are fairly common; and often the functions are rather simple in form, especially for very large systems. It is easier to take advantage of these possibilities with analytic differentiation than numerical differentiation with the consequence that although the former may be more trouble to set up, it will prove very much cheaper. Those concerned with application packages should be especially alert to this possibility because of the restriction to a special class of equations in their area. An example is the package [7] in which the user describes his chemical kinetics problem in a manner natural to him as a chemist. The package sets up the differential equations and forms the (easy) Jacobian analytically on its own. At some computing installations symbol manipulation languages furnish a quite practical tool for the generation of Jacobians. Depending on the problem and the efficiency of the code generated by the

symbolic processor, the analytical derivatives may, or may not, be evaluated more cheaply than by differencing.

One of the reasons for preferring analytical Jacobians is that scaling difficulties in differencing can lead to poor numerical Jacobians. Good differencing algorithms try to cope with this sort of trouble and, of course, the codes will work with poor Jacobians anyway, but the net effect is that scaling troubles increase the cost and decrease the reliability of differencing as compared to analytical Jacobians.

An intermediate procedure is for the user to provide structure information which the code then uses to construct Jacobians by differencing. A trick discovered by Curtis, Powell, and Reid [16] may achieve substantial reductions in the cost of differencing because of known structure. A particularly simple and quite common case is that of a banded system. We say the system is banded, with half bandwidth m, if for each i, the i-th equation does not involve the variables y_j for $|j - i| > m$. For such a system the Jacobian has non-zero elements only within a band about the main diagonal, whence the name. As it turns out, a Jacobian of half bandwidth m can be formed by differencing in only $2m + 1$ extra evaluations of f regardless of the size of the system of equations. Clearly this leads to large reductions in machine time even when m is not particularly small compared to n.

Storage can be a limiting factor in the solution of large stiff problems because one must store the Jacobian or a closely related matrix for which one must solve a linear system of evaluations. A major disadvantage of the implicit Runge-Kutta schemes is that the size of the nonlinear system to be solved at each step is a multiple of the

number of equations, the higher the order, the bigger the multiple. Because the Jacobian grows like the square of the number of unknowns, for even moderate sized systems the order in the code COLODE [10] has to be restricted severely to hold the program in rapid memory.

The repeated solution of linear systems at each step is a relatively expensive part of the solution process so we must consider how to do it efficiently subject to the constraint of available storage. Because the matrices involved depend on the step size, it is particularly easy to see with extrapolation procedures that one does a lot of linear algebra. Assuming that the same approximate Jacobian will serve for the whole step, one repeatedly advances a method such as the backward Euler method through this interval with successively smaller fractions of the basic step size and combines these results at the end of the basic step. Each sweep involves a different linear system which must be solved repeatedly at each fractional step of the sweep.

For small to moderate sized systems one can use elimination to factor the matrix and then use these factors to do the necessary iterations efficiently. To solve medium to large problems we must again resort to structural information. Band structure is the easiest to accomodate. Because the factors of a band matrix are also of band form it is possible to compute and store only those elements known to be potentially non-zero. If the band width is relatively small, $m << n$, one greatly reduces the storage and greatly increases the efficiency of the solution of the linear system by taking advantage of the band form. More generally a matrix is said to be *sparse* if most of its elements are zero. This is typical of many very large stiff problems. There are schemes for storing only the

non-zero elements of sparse matrices and for doing elimination in such a way as to introduce relatively few non-zero elements. Some storage is required for the schemes themselves and some extra effort in the elimination, so they do not begin to pay off in either storage or speed until only a few percent of the entries in the matrices are non-zeros. Fortunately this is often the case. Problems have been solved which are so large that iterative methods had to be used for the linear systems because such methods require less storage than a factorization. This is an active field of research at the present and we may well see iterative techniques being used for less special problems.

Since it is often true that evaluation of the function and even the Jacobian are not particularly expensive for large problems, the extensive linear algebra and data transfers are often a large fraction of the total cost. FORTRAN does not take full advantage of the overlapping of data transfers and computation and of hardware possibilities of pipeline and vector machines. FORTRAN callable assembly language modules are becoming available which allow more efficient handling of basic tasks than are possible in the FORTRAN codes being widely disseminated. The report [18] gives some codes and timing comparisons on a CDC 7600 of routines for the LU decomposition of a matrix and the forward and backward substitution process for solving a linear system. For only three equations, the FORTRAN programs are slightly faster because of some additional overhead in the linear algebra modules, but the more specialized routines quickly pull ahead and show an improvement of a factor of about three for 200 equations. (The solving routine benefits a little more than

the decomposition routine.) For large problems there are clearly important cost savings which can be achieved on some machines in this manner.

Many problems lead to implicit sets of differential equations such as

$$M\,y' = K\,y + p(x) \tag{4.1}$$

where the matrices M and K may depend on y, x, and even y'. It is not necessary, nor even desirable, to invert these to the explicit form

$$y' = M^{-1}[K\,y + p(x)] \tag{4.2}$$

when a quasi-Newton iteration is used for the corrector. If, for example, the backward Euler method is used as an integrator, it can be substituted in

$$F(y,\, y',\, x) = 0 \tag{4.3}$$

at $x = x_{n+1}$ to eliminate y'_{n+1} and get

$$F(y_{n+1}\,,\ \frac{y_{n+1} - y_n}{h}\,,\ x_{n+1}) = 0$$

which is to be solved for y_{n+1}. Dealing with the implicit equation (4.3) directly can save arithmetic and storage in large sparse problems. This is clear from equations (4.1) and (4.2) when M and K are constant. Whereas the Jacobian of (4.1) will be K - M/h which will be sparse, the Jacobian of (4.2) is $M^{-1}(K - I/h)$ which will normally be dense.

A very important part of any code for solving differential equations, whether stiff or non-stiff, is automatic control of the step size and order of method used. While the basic strategy is straightforward -- the two are chosen to try and minimize the amount of work done to integrate over the interval, the implementation is not. The problem lies in the fact that there is not yet an adequate theory to tell us how

to choose these parameters, so the choice is based on the extension of existing theory to situations in which it probably does not apply, coupled with a lot of testing and tuning of codes to make them as reliable as possible. Because of this, it is possible for a person to implement a set of formulas into a code and use the same basic step and order control strategy as another code, and yet have a code that is orders of magnitude slower on some difficult problems. (This is illustrated in [1] which compares a number of codes for non-stiff problems, including four based on Adams methods and rather similar step and order control strategies.) The lesson to be learned from this is that, whenever possible, an existing piece of well tested and well documented software should be used, and if possible, it should be used without change.

Variable order codes adapt the formula used to the observed characteristics of the solution and have proved very efficient in general use. One should appreciate that such codes are relatively inefficient during the initial stages of the computation while they find an appropriate order. Ordinarily this is an unimportant part of the overall expense, but there are exceptions. We have seen examples in flame chemistry where one has partial differential equations describing the gas flow coupled to ordinary differential equations describing reactions taking place in the flow. A pseudo steady-state approximation is made in which one holds the gas flow parameters constant for a time period during which one integrates the ordinary differential equations. Using these values one re-solves the partial differential equations for the time zone or advances into the next time zone - the details do not matter here. The point is that at the end of each zone, the ordinary differential

equations change. Codes based on backward differentiation formulas prove very inefficient because they must be re-started at each zone. Things do not change much from zone to zone (else they would be shortened) so that the last step size used in one zone would be reasonable in the next if one retained the same formula. Fixed order methods appear to offer considerable advantage in this situation and the limited experience agrees. To some degree, extrapolation methods which permit high orders must also be at a disadvantage because the order and the length of the step will be held back by the length of the zone. A low order and small step size can be obtained more cheaply by a code with fixed order.

5. SOLVING A DIFFERENTIAL EQUATION

There are a number of meanings of "solving" a differential equation numerically. Sometimes one wants an approximate solution at a single point. More often a solution is desired on a set of points so as to get a table of the solution. Other times one wants either a continuous approximate solution curve or a table so dense as to be equivalent. It is important to realize that where and how frequently one wants output points can have a serious effect on the cost. The most efficient action depends on the method implemented, the principal factor being how far, relatively speaking, the code steps before producing an approximate solution. For example, high order implicit Runge-Kutta schemes and extrapolation schemes advance very much further in a step than does a code based on the backward differentiation formulas. The former produce output at a given point by stepping so as to actually hit the point. Requesting output more frequently than the natural step size chosen by the code will severely degrade the efficiency of such methods for a variety of reasons. If one seeks an answer at only a few points, especially if he is willing to accept the natural output points, output is not a problem. Some methods, like the backward differentiation formulas, provide continuous polynomial solutions which can be evaluated at negligible cost. Not all codes implement this equally well. In this matter one needs to consider what is required in the way of output and how it impacts the repertoire of codes.

The user's meaning of accuracy can affect the results considerably. One common scheme is to measure the error relative to the maximum (absolute) value of the solution component seen so far in the integration. Another common scheme is a mixture of absolute error and error relative to the

solution magnitude. It is important to be able to specify error tolerances for each component of the solution because scaling of components often differ radically for stiff problems. Stiff problems almost always involve transients during which the solution changes sharply. In the survey [19] about half the users required the accurate solution of these transients as well as that of the slowly varying portions. To do this one must use a suitable error control and the net effect is that in the transient, accuracy dominates the choice of step size rather than stability. As a result the transients become a relatively expensive part of the integration for most codes. One may need to follow the transients with some accuracy to assure that the proper equilibrium solution is picked up but generally, if accurate transients are not needed, one should not try to get them.

Stiff problems are relatively expensive to solve and the expense depends much more strongly on the tolerance than is true of the best codes for non-stiff problems. The physical origin of stiff problems rarely makes high accuracy meaningful because fundamental quantities are known inaccurately. The case of semi-discretization of partial differential equations is an easily understood and important case in point. If the spacial discretization is crude, it makes no sense to solve the ordinary differential equations very accurately at all. In the survey [19] accuracies of one or two digits were by far the most common requests. An accuracy of five digits was considered stringent. Apparently experience says that accuracies in this general area represent a bearable expense with our currently available codes and machines.

REFERENCES

[1] Shampine, L. F., Watts, H. A., and Davenport, S. M., "Solving non-stiff ordinary differential equations -- the state of the art," SIAM Review, 18, #3, pp. 376-411, 1976.

[2] Shampine, L. F., "Stiffness and non-stiff differential equation solvers," in Collatz, L., ed. Numerische Behandlung von Differentialgleichungen, ISNM 27, Birkhauser, Basel, pp. 287-301, 1975.

[3] Shampine, L. F., and Gordon, M. K., Computer Solution of Ordinary Differential Equations: The Initial Value Problem, W. H. Freeman and Co., San Francisco, 1975.

[4] Bjurel, G., Dahlquist, G., Lindberg, B., Linde, S., and Oden, L., "Survey of Stiff Ordinary Differential Equations," Report NA 70.11, Dept. of Information Processing, Royal Institute of Technology, Stockholm, 1970.

[5] Shampine, L. F., "Stiffness and non-stiff differential equation solvers II: detecting stiffness with Runge-Kutta methods," ACM Trans. on Math. Software, to appear.

[6] Gear, C. W., "Estimation of Errors and Derivatives in Ordinary Differential Equations," Information Processing, 3, pp. 447-451, 1974.

[7] Joyce, D. C., "Survey of extrapolation processes in numerical analysis," SIAM Review, 13, pp. 435-488, 1971.

[8] Schryer, N. L., "An extrapolation step size monitor for solving ordinary differential equations," Proceedings of ACM, pp. 140-148, 1974.

[9] Lindberg, B., "A stiff system package based on the implicit midpoint method with smoothing and extrapolation," in R. A. Willoughby, ed., Stiff Differential Systems, Plenum Press, New York, pp. 201-215, 1974.

[10] Hulme, B. L,, "COLODE: A colocation subroutine for ordinary differential equations," Report SAND74-0380, Sandia Laboratories Albuquerque, New Mexico, 1974.

[11] Gear, C. W., "The Automatic Integration of Stiff Ordinary Differential Equations," Information Processing, pp. 187-193, 1969.

[12] Gear, C. W., "Algorithm 407: DIFSUB for Solution of Ordinary Differential Equations [D2]," Communications of the ACM, Vol. 14, 3, pp. 185-190, 1971.

[13] Hindmarsh, A. C., "GEAR: Ordinary differential equation solver," Report UCID-30001, Lawrence Livermore Laboratory, Livermore, CA, 1972.

[14] Skeel, R. D., and Kong, A. K., "Blended Linear Multistep Methods," Report UIUCDCS-R-76-800, Dept. of Computer Science, University of Illinois, Urbana, IL, 1976.

[15] Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.

[16] Curtis, A. R., Powell, M. J. D., and Reid, J. K., "On the estimation of sparse Jacobian matrices," J. Inst. Maths. Applics. 13, pp. 117-119, 1974.

[17] Edsberg, L., "Integration package for chemical kinetics," in R. A. Willoughby, ed., Stiff Differential Systems, Plenum Press, NY, pp. 81-94, 1974.

[18] Hindmarsh, A. C., Sloan, L. J., Fong, K. W., and Rodrigue, G. H., "DEC/SOL: Solution of dense systems of linear algebraic equations," Report UCID-30137, Lawrence Livermore Laboratory, Livermore, CA, 1976.

[19] Shampine, L. F., and Gordon, M. K., "Typical problems for stiff differential equations," SIGNUM Newsletter, 10, p. 41, 1975.

Form AEC–427
(6/68)
AECM 3201

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.
COO-2383-0036

2. TITLE
A User's View of Solving Stiff Ordinary Differential Equations

3. TYPE OF DOCUMENT (Check one):

[XX] a. Scientific and technical report
[ ] b. Conference paper not to be published in a journal:
Title of conference _____
Date of conference _____
Exact location of conference _____
Sponsoring organizetion _____
[ ] c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

[XX] a. AEC's normal announcement and distribution procedures may be followed.
[ ] b. Make aveilable only within AEC and to AEC contractors and other U.S. Government agencies end their contractors.
[ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear
Professor and Principal Investigator

Organization

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Signature

Date
October, 1976

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

[ ] a. AEC patent clearance has been grented by responsible AEC patent group.
[ ] b. Report has been sent to responsible AEC patent group for cleerance.
[ ] c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-829	2.	3. Recipient's Accession No.

4. Title and Subtitle

A User's View of Solving Stiff Ordinary Differential Equations

5. Report Date: September, 1976

6.

7. Author(s)
L. F. Shampine, C. W. Gear

8. Performing Organization Rept. No. UIUCDCS-R-76-829

9. Performing Organization Name and Address
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
US ERDA E(11-1) 2383

12. Sponsoring Organization Name and Address
U. S. Energy Research and Development Administration
Chicago Operations Office
9800 South Cass Avenue
Argonne, IL 60439

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This paper aims to assist the person who needs to solve stiff ordinary differential equations.

First we identify the problem area and the basic difficulty by responding to some fundamental questions: Why is it worthwhile to distinguish a special class of problems termed "stiff"? What are stiff problems? Where do they arise? How can we recognize them?

Second we describe the characteristics shared by methods for the numerical solution of stiff problems. These characteristics have important implications as to the convenience and efficiency of solution of even routine problems. Understanding them is indispensable to the assembling of codes for the very efficient solution of special problems or for solving exceptionally large problems at all.

Third we shall briefly discuss what is meant by "solving" a differential equation numerically and what might be reasonably expected in the case of stiff problems.

17. Key Words and Document Analysis. 17a. Descriptors

stiff equations
numerical solution of ordinary differential equations

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited	UNCLASSIFIED	35
	20. Security Class (This Page) UNCLASSIFIED	22. Price

FEB 9 1977

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.824-829(1978
Internal report /
3 0112 088402992